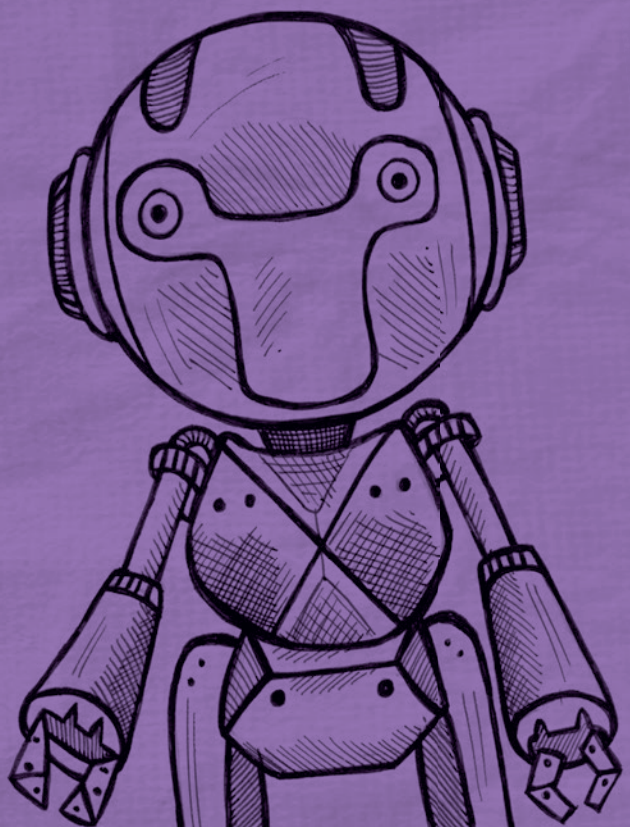
endorsed for edexcel

REVISE EDEXCEL GCSE (9–1)

Computer Science

REVISION GUIDE



For the 9–1 exams

Includes free online edition

Pearson

REVISE EDEXCEL GCSE (9–1)
Computer Science

REVISION GUIDE

Series Consultant: Harry Smith

Author: David Waller

A note from the publisher

In order to ensure that this resource offers high-quality support for the associated Pearson qualification, it has been through a review process by the awarding body. This process confirms that this resource fully covers the teaching and learning content of the specification or part of a specification at which it is aimed. It also confirms that it demonstrates an appropriate balance between the development of subject skills, knowledge and understanding, in addition to preparation for assessment.

Endorsement does not cover any guidance on assessment activities or processes (e.g. practice questions or advice on how to answer assessment questions), included in the resource nor does it prescribe any particular approach to the teaching or delivery of a related course.

While the publishers have made every attempt to ensure that advice on the qualification and its assessment is accurate, the official specification and associated assessment guidance materials are the only authoritative source of information and should always be referred to for definitive guidance.

Pearson examiners have not contributed to any sections in this resource relevant to examination papers for which they have responsibility.

Examiners will not use endorsed resources as a source of material for any assessment set by Pearson.

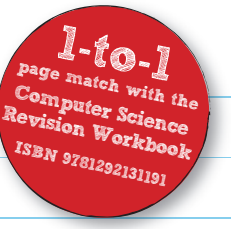
Endorsement of a resource does not mean that the resource is required to achieve this Pearson qualification, nor does it mean that it is the only suitable material available to support the qualification, and any resource lists produced by the awarding body shall include this and other appropriate resources.

Question difficulty

Look at this scale next to each exaam-style question. It tells you how difficult the question is.

Pearson

Contents



A small bit of small print

Edexcel publishes Sample Assessment Material and the Specification on its website. This is the official content and this book should be used in conjunction with it. The questions in *Now try this* have been written to help you practise every topic in the book. Remember: the real exam questions may not look like this.

Algorithms

Algorithms provide precise instructions needed to solve a problem. All computer programs are algorithms. An algorithm is a step-by-step procedure for solving a problem or carrying out a task.

Uses for algorithms

Algorithms are often used to improve efficiency by removing the need for human input, for example, share trading on the stock market. A computer following an algorithm can decide which deal to make far more quickly than a human and a split second difference can be worth millions of pounds.

Constructs of an algorithm

There are three constructs used in an algorithm – **sequence**, **selection** and **iteration**. Here is an algorithm for a guessing game.

1. Person A thinks of a number between 1 and 20.
2. Person B makes a guess.
3. If the guess is too great:
   (a) person A says 'Too high'
   (b) go to step 2.
4. If the guess is too small:
   (a) person A says 'Too low'
   (b) go to step 2.
5. If the guess is correct:
   (a) person A says 'Correct'.

Sequence – instructions need to be given in the correct order for the game to be played successfully.

Selection – decisions have to be made and a course of action selected.

Iteration – previous steps are repeated until there is a desired outcome (in this case, until there is a correct guess).

Worked example

Write an algorithm to make a cup of instant coffee. It should be annotated to show where sequence, selection and iteration have been used. **(5 marks)**

1 Fill kettle with water. (sequence)
2 Turn on kettle. (sequence)
3 Place coffee in the cup. (sequence)
4 Check if the water is boiling. (selection)
5 If water is not boiling go back to step 4. (iteration)
6 Pour water into the cup. (sequence)
7 If needed, add milk and sugar. (selection)
8 Stir. (sequence)

Selection occurs when a decision has to be made and iteration is used when a sequence of activities has to be repeated.

Reusing algorithms

Once an algorithm has been written, it can be reused with slight changes for solving similar problems – which is much quicker than starting from scratch each time.

Now try this

Algorithms use selection and iteration.

(a) State what is meant by selection. **(1 mark)**

(b) State what is meant by iteration. **(1 mark)**

Had a look ☐ Nearly there ☐ Nailed it! ☐

Algorithms: pseudo-code

Algorithms can be displayed using pseudo-code.

Pseudo-code

- Pseudo-code is a way of expressing an algorithm in structured English that resembles computer language.
- There are many different varieties of pseudo-code but, in your exam, questions will be written using the Edexcel version.
- You may provide answers in any style of pseudo-code as long as the meaning could be reasonably understood by a competent programmer.

Use of pseudo-code

- Pseudo-code uses commands, keywords and structures similar to those found in computer languages.
- Pseudo-code cannot be understood by computers, but is used to develop the **logic** of the algorithm without having to bother about **syntax** (the rules of grammar such as spelling or punctuation).
- A human can follow the logic of an algorithm even if there are spelling mistakes or missing brackets but a computer cannot execute code if there are similar syntax errors.
- A solution in pseudo-code is converted into a programming language such as Python or Java.

Example of pseudo-code

This pseudo-code shows an algorithm for calculating the cost of sending parcels. The pseudo-code shows the logic of the algorithm.

```
SET parcel TO 'y'
WHILE parcel = 'y' DO # The loop will run while parcel = 'y'
RECEIVE weight FROM (INTEGER) KEYBOARD
  IF weight <= 2 THEN
    SET cost TO 2 # Parcels up to 2kg cost £2
  ELSE
    SET excess TO weight - 2
    SET cost TO 2 + excess*3 # Over 2kg the cost is £3 per kg
  END IF
  SEND cost TO PRINTER
  SEND 'Press 'y' to process another parcel.' TO DISPLAY
  RECEIVE parcel FROM (STRING) KEYBOARD
END WHILE
```

Variables have been used, e.g. `parcel`, `weight` and `excess`.

Indefinite iteration using a `while` loop as the number of parcels to be processed is not known at the start.

See page 20 for more on WHILE loops.

Comments are included in pseudo-code using the # symbol. Comments are used by programmers to help others to follow their logic, and to act as a reminder of the logic. It is good practice to include comments.

Indenting helps with the logic of the algorithm by showing dependency – everything inside the `ELSE` block (lines 7 and 8) is indented to show the lines dependent on this instruction. The pseudo-code is not going to be executed by a computer, so indentation is not strictly necessary but it makes the code easier to understand.

The user is asked if they want the loop to run again. It will stop if the input is other than `y`.

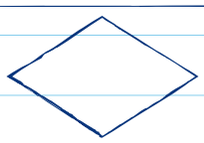
Now try this

Write an algorithm that will find the sum of ten numbers entered by a user and output the result. Use pseudo-code. Include comments to explain how the code works. **(6 marks)**

Algorithms: flowcharts

Algorithms can be displayed as flowcharts (also known as flow diagrams).

Symbols used in flowcharts

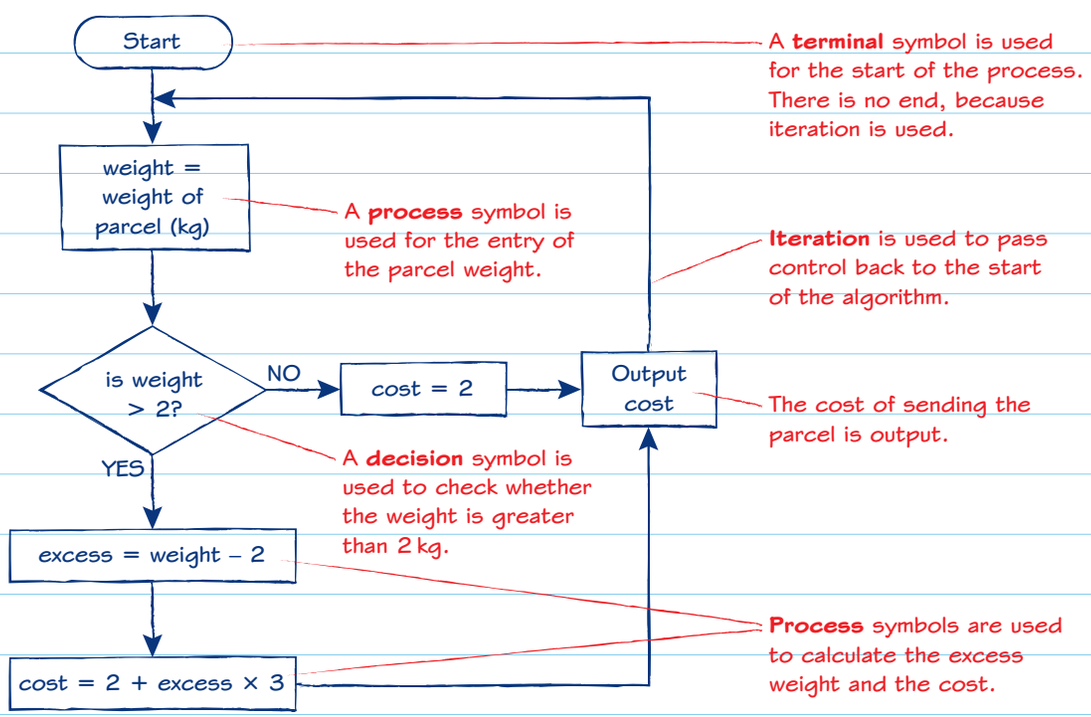
				
Terminal Shows start and end of an algorithm.	**Decision/ Selection** Shows yes/ no or true/ false decisions where there are two possible outcomes.	**Process** Shows data processing, e.g. a calculation.	**Subroutine** Shows a function or procedure that has its own flowchart.	**Line** Represents control passing between the other symbols.

Flowcharts in use

The cost of sending a parcel by courier varies according to the weight of the parcel:

- if a parcel weighs <= 2 kg, the cost is £2
- if a parcel weighs > 2 kg, it costs £2 + £3 × excess weight.

This flowchart represents an algorithm for calculating the cost of sending parcels.



Now try this

You are expected to use the correct notation for flowcharts.

Draw a flowchart for an algorithm to find the sum of 10 numbers and output the result. You do not have to use the input/output symbol in your answer but you will not be penalised if you do. **(5 marks)**

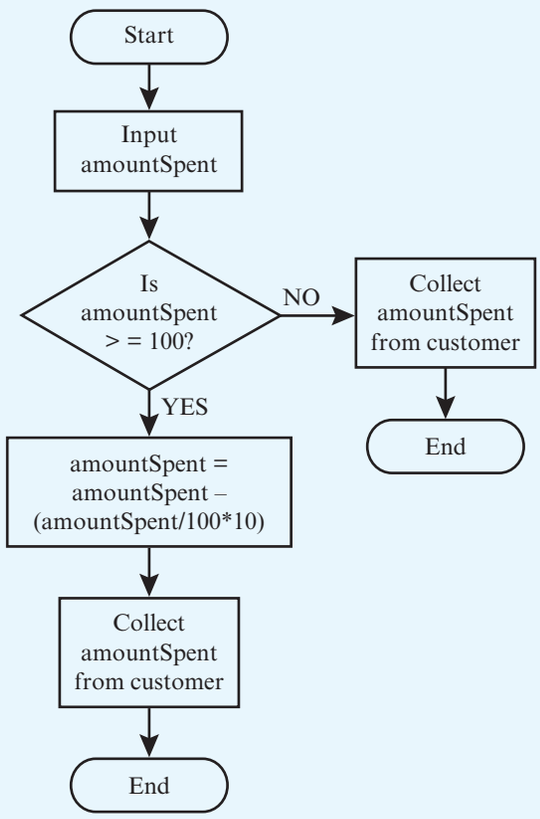
Purpose of an algorithm

Understanding algorithms

You should be able to write and interpret algorithms expressed as written descriptions and program code, as well as flowcharts and pseudo-code and understand their purpose.

Worked example

Examine this flowchart.



(a) State the function of this algorithm. **(2 marks)**

The function of the algorithm is to check the amount a customer has spent.

If it is more than or equal to £100, then they are given a discount.

The amount they have to pay minus the discount is calculated.

(b) (i) State the minimum amount a customer has to spend to receive a discount. **(1 mark)**

£100

(ii) State the percentage discount a customer receives if they spend the minimum amount or more. **(1 mark)**

10%

(c) Calculate how much a customer would have to pay if the amount spent was £150. **(2 marks)**

150 – (150/100*10) = 150 – 15
= £135.00

Now try this

Study this pseudo-code algorithm and then answer the following questions.

```
SEND 'Please enter the three numbers.'
TO DISPLAY
RECEIVE number1 FROM (INTEGER) KEYBOARD
RECEIVE number2 FROM (INTEGER) KEYBOARD
RECEIVE number3 FROM (INTEGER) KEYBOARD
IF number1 > number2 THEN
  IF number1 > number 3 THEN
    SEND number1 TO DISPLAY
  ELSE
    SEND number3 TO DISPLAY
  END IF
ELSE
  IF number2 > number3 THEN
    SEND number2 TO DISPLAY
  ELSE
    Send number3 TO DISPLAY
  END IF
END IF
```

(a) State the purpose of this algorithm. **(1 mark)**

(b) Explain how the algorithm accomplishes this purpose. **(3 marks)**

Completing algorithms

You may be asked to complete an algorithm in the exam.

Worked example

A student is monitoring the temperature in a greenhouse three times a day for one week. She has stored the temperatures in a 2-dimensional array named 'temperatures' and is creating an algorithm to find and output the average temperature for each day and output the average for the week.

The following shows part of her algorithm written in pseudo-code.

Complete the algorithm by completing or inserting the missing lines of code. **(4 marks)**

```
1 SET week TO 0
2 FOR day
3   SET total TO 0
4   FOR temp FROM 0 TO 2 DO
5     SET total TO
6
7   SET average TO total/3
8   SET week TO
9   SEND 'The average for day
10
11  SET weekAverage TO week/7
12  SEND '
```

Carefully read the scenario and the code the student has written. For example, there are 3 readings per day for 7 days. You will need to know this when setting up the FOR loops.

Draw a table or matrix to help you visualise how the data is stored.

	0	1	2
0	0, 0	0, 1	0, 2
1			
2			
3	3, 0	3, 1	3, 2
4			
5			
6			6, 2

This will help with the loops that have to be set up.

```
1 SET week TO 0
2 FOR day FROM 0 TO 6 DO
3   SET total TO 0
4   FOR temp FROM 0 TO 2 DO
5   SET total TO total + temperatures[day, temp]
6   END FOR
7   SET average TO total/3
8   SET week TO week + average
9   SEND 'The average for day' & day &
    ' is ' & average & '.' TO DISPLAY
10  END FOR
11  SET weekAverage TO week/7
12  SEND 'The average for the week is ' &
    weekAverage & '.' TO DISPLAY
```

Now try this

Complete the following algorithm which a student created for a guessing game.

```
1 SET mysteryNumber TO RANDOM(10)
2 SEND 'Please enter a number between 1 and 10.' TO DISPLAY
3 RECEIVE number
4
5   SEND 'Too high' TO DISPLAY
6
7
8         SEND 'Too low' TO DISPLAY
9
10        SEND 'You guessed it!' TO DISPLAY
11    END IF
12
```

(4 marks)

Had a look ☐ Nearly there ☐ Nailed it! ☐

Interpreting correct output

It is often difficult to look at an algorithm produced by another programmer and understand what it does and how it does it.

Dry runs

A dry run is a method used to investigate the **functioning** of an algorithm. It is a mental or pencil and paper run-through of an algorithm that does one step at a time. The following are examined for each step:

- the inputs needed
- the data processing
- the outputs.

A dry run can help you to understand an algorithm and also to find any errors.

Trace tables

While the dry run is being worked through, it is helpful to use a trace table in which to write down the values of each variable, input and output and how they change as the program is running. A trace table has columns for each of the variables and rows for each of the steps in the algorithm.

An algorithm and trace table

A student has written an algorithm to carry out a linear search that outputs `'Found'` if a particular search item is found in a given list. She tests the algorithm with the following data: 3, 6, 9, 13, 17, 21

She uses the number 13 as the search item.

The trace table below shows the values of the different variables compared with the search item.

```
SET list TO [3, 6, 9, 13, 17, 21]
SET found TO False
RECEIVE item FROM (INTEGER) KEYBOARD
FOR index FROM 0 TO LENGTH (list) - 1 DO
  IF item = list[index] THEN
    SET found TO True
    SEND 'Found' TO DISPLAY
  END IF
END FOR
```

The variable `found` contains the value `false` at the start of the algorithm and is changed to `true` if the search item is found.

Columns are used for the variables and the output.

The variable `item` has the value 13 throughout the running of the algorithm.

item	found	index	list[index]	output
13	false	0	3	
	false	1	6	
	false	2	9	
	true	3	13	Found
	true	4	17	
	true	5	21	

The output when the search item is found.

The variable `index` increments from 0 to 5, as the length of the array is equal to 6.

The data items at each index position.

Now try this

Here is an algorithm expressed in pseudo-code.

```
SET number TO 3
FOR index FROM 1 TO 3 DO
  SET number TO number + index
  SEND number TO DISPLAY
NEXT FOR
```

Complete the trace table to show the execution of the algorithm.

You may not need to fill in all the rows in the table.

number	index	output

(4 marks)

Using trace tables

Trace tables have columns for each of the variables and input and output values.

Worked example

Investigate the following algorithm and complete a trace table using these values.
At the start of the following algorithm, firstNumber = 2 and secondNumber = 3

```
SET total TO firstNumber + secondNumber
WHILE firstNumber < total DO
  SET firstNumber TO firstNumber + 1
  SET secondNumber TO secondNumber + firstNumber
END WHILE
SEND secondNumber TO DISPLAY
```

(4 marks)

total	firstNumber	secondNumber	output
5	3	6	

A table with columns for firstNumber, secondNumber, total and the output is needed. The variables are shown in the order in which they appear in the algorithm.

total	firstNumber	secondNumber	output
5	3	6	

In the first iteration, total = 5 (2 + 3) and, as firstNumber is less than total (2 < 5), both firstNumber and secondNumber will be incremented as instructed in the algorithm.

total	firstNumber	secondNumber	output
5	3	6	
5	4	10	

In the next iteration, total is still equal to 5 as it is not incremented in the loop. firstNumber is still less than total (3 < 5), so firstNumber and secondNumber are both incremented as before.

total	firstNumber	secondNumber	output
5	3	6	
5	4	10	
5	5	15	

firstNumber is still less than total so there will be another iteration.

The WHILE loop will run while firstNumber is less than total, total is calculated before the loop starts and so will never change and is equal to the sum of the starting values of firstNumber and secondNumber. On the last turn of the loop (when firstNumber = 4), it will be incremented to 5 and will then stop.

total	firstNumber	secondNumber	output
5	3	6	
5	4	10	
5	5	15	
			15

firstNumber is now equal to total so the program execution will break out of the loop. The value of secondNumber will be printed.

Now try this

Here is an algorithm expressed in pseudo-code.

```
SET number TO 3
SEND number TO DISPLAY
FOR index FROM 1 TO 3 DO
  SET number TO number + 6
  SEND number TO DISPLAY
END FOR
```

Complete the trace table to show the execution of the algorithm. You may not need to fill in all the rows in the table.

number	index	output

(4 marks)

Had a look ☐ Nearly there ☐ Nailed it! ☐

Identifying and correcting errors

Identifying and correcting errors shows that you understand and can interpret algorithms.

Worked example

A student has created an algorithm to calculate the average mark they have achieved in Computer Science.

```
1   SET total TO 0
2   SET numberEntered TO 1
3   SET anotherEntry TO 'N'
4   WHILE anotherEntry = 'Y' DO
5     RECEIVE mark FROM (INTEGER) KEYBOARD
6     SET total TO mark
7     SET numberEntered TO numberEntered - 1
8     SEND 'Do you want to enter another mark (Y or N)?' TO DISPLAY
9     RECEIVE anotherEntry FROM (CHARACTER) KEYBOARD
10  END WHILE
11  SET average TO total / numberEntered
12  SEND The average is ' & average & ' marks.' TO DISPLAY
```

When checking for errors you must read through the algorithm very carefully. Check that:

- variables have been initialised to the correct values
- loops have been set up correctly
- speech marks and opening and closing brackets match.

There are **five** errors in this algorithm.
Some are logic errors and some are syntax errors.
Identify and correct the errors. **(5 marks)**

```
2   SET numberEntered TO 0
3   SET anotherEntry TO 'Y'
6     SET total TO total + mark
7     SET numberEntered TO numberEntered + 1
12  SEND 'The average is ' & average & ' marks.'TO DISPLAY
```

Pseudo-code doesn't normally have numbered lines but, in this case, it makes it easier for candidates to answer the question by simply writing relevant lines of code rather than having to write out the whole algorithm.

Now try this

The following is an algorithm for a game where a user has to guess a random number.

```
1   SET randomNumber TO RANDOM(10)
2   SET correct TO False
3   SET goes TO 0
4   WHILE correct  False DO
5     RECEIVE guess FROM (INTEGER) KEYBOARD
6     SET goes TO goes + 1
7     IF guess = randomNumber
8       SET correct TO True
9     END IF
10    IF guess > randomNumber THEN
11      SEND 'Too low.' TO DISPLAY
12    END IF
13    IF guess > randomNumber THEN
14      SEND 'Too high.' TO DISPLAY
15    END IF
16  END
17  SEND 'You had ' & ' goes ' & guesses.'
```

There are **five** errors in this algorithm. Some are logic errors and some are syntax errors.
Identify and correct the errors. **(5 marks)**

Linear search

When large amounts of data are searched, it is essential that the searching algorithm is as efficient as possible. Standard search algorithms include **linear** and **binary** searches.

Linear search

A linear search is **sequential**. This algorithm starts at the beginning of the list and moves through item by item until it finds the matching item or reaches the end of the list. To find the number 37 in a list, a linear search would start at the first entry (20) and simply move to the next item until it finds 37 (or reaches the end of the list).

indices

0	1	2	3	4	5	6	7	8
20	35	37	40	45	50	51	55	67
↑	↑	↑						
≠	≠	=						

The number is found on the third comparison at index 2.
Remember: indices start at 0.

Brute force

A linear search is an example of a brute-force algorithm. It does not use any specialist techniques, only raw computing power. It is not an efficient method as each search starts at the beginning and keeps going until the item is found or the end of the list is reached.

The efficiency of a linear search

If there is a list of 500 items:

- In the **best case**, the search item is the first item found.
- In the **worst case**, the search item is the last item found – number 500.

The smaller the list the more efficient the linear search.

Average case: $\frac{(500 + 1)}{2} = 250.5$

Worked example

Describe in words an algorithm for carrying out a linear search. **(6 marks)**

1 If the length of the list is zero, stop.
2 Start at the beginning of the list.
3 Compare the list item with the search criterion.
4 If they are the same, then stop.
5 If they are not the same, then move to the next item.
6 Repeat steps 3 to 5 until the end of the list is reached.

It is best to number the steps in case iteration is needed. In this answer, the step numbers are used for the iteration command in step 6.

Remember that the length of the list must be checked to ensure that there are some items to search (step 1).

When coding this algorithm a Boolean (see pages 21 and 57) flag would be set to 'yes' when the item was found so that no further searching would take place.

Now try this

Find out more about arrays on page 23.

Write an algorithm expressed in pseudo-code to ask a user for a search item and carry out a linear search on data stored in an array to find that item. **(6 marks)**

Had a look ☐ Nearly there ☐ Nailed it! ☐

Binary search

A binary search compares the search item with the median item in a list, repeatedly splitting the list in half until the item is found or there are no more items left to search.

Binary search

To use a binary search, the list must have already been sorted into ascending order. A binary search uses a divide-and-conquer strategy to increase its efficiency:

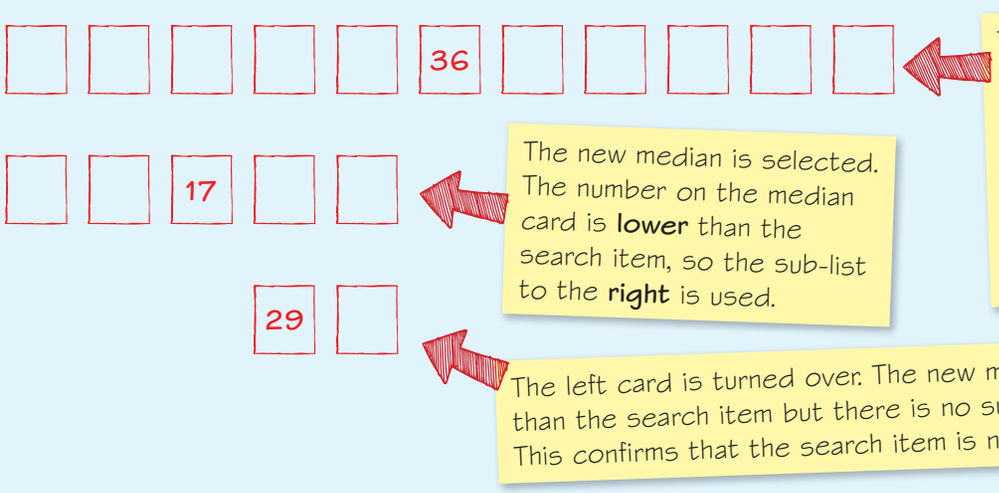
- select the median
- compare it with the search item
- if the search item is **lower**, discard the median and the **higher items**
- if the search item is **higher**, discard the median and **lower items**
- recalculate the new median
- repeat this process until the search item is found (or is not found) in the list.

Median values

- The **median** is the middle item in a list. In a list of 13 items, the median item is the 7th item in the sorted list.
- If there is an odd number of items in the list (e.g. 13) then the median is the 7th number but for an even number of items (e.g. 10) then the median would be 5.5 – midway between 5 and 6. The lower number is used.
  It can be found using the formula:
  `Median = (length of list + 1) DIV 2`
- Lists of numbers need to be sorted in numerical order.
- Lists of words or text strings need to be sorted into alphabetical order.

Worked example

Numbers have been written onto 11 cards that have then been placed face down in ascending order.
Show the stages in a binary search to see if the number 28 is on one of the cards. **(4 marks)**



The median card is selected and turned over. The number on the median card is **higher** than the search item, so the sub-list to the **left** is used.

The new median is selected. The number on the median card is **lower** than the search item, so the sub-list to the **right** is used.

The left card is turned over. The new median is higher than the search item but there is no sub-list to the left. This confirms that the search item is not in the list.

The search has confirmed that the number 28 is not on any of the cards.

Now try this

Describe the stages in applying a binary search to the following list to find the number 17.
3, 5, 9, 14, 17, 21, 27, 31, 35, 37, 39, 40, 42 **(4 marks)**

Comparing linear and binary searches

The binary search algorithm is more efficient than a linear search but the list has to be sorted into ascending or descending order.

The efficiency of linear vs binary search

The table shows a comparison for a list of 500 items.

	Linear search		Binary search	
		Number of selections		Number of selections
Best case	The search item is the first one.	1	The search item is the median.	1
Worst case	The search item is the last one: number 500	500	Assuming that the median was too large each time, the following would be selected: 250, 125, 63, 32, 16, 8, 4, 2, 1	9
Average case	$\frac{(500 + 1)}{2} = 250.5$ That would mean 251 searches.	251	$\frac{(9 + 1)}{2} = 5.0$ That would mean 5 searches.	5

Worked example

Describe **one** benefit and **one** drawback of using a binary search rather than a linear search.

(4 marks)

A benefit of the binary search is that it uses a strategy to minimise the number of comparisons that are made and is therefore more efficient than a linear search when there are a lot of items in the list.

A drawback of using the binary search is that the data must first be sorted into ascending order. Sorting the data will take time and reduce the overall efficiency.

A binary search uses a strategy to reduce the number of comparisons needed but a linear search starts at the first item and checks every one until the item is found or it gets to the end of the list. A linear search is more efficient for small numbers of items, as the list does not have to be sorted.

Now try this

A student has the following names of friends stored in a list.

Ahmad	Ann	Claire	Denzil	Mary	Matt	Paru	Stephen	Zoe

Show the stages of a binary search to find the name Ann in the above data. **(2 marks)**

Had a look ☐ Nearly there ☐ Nailed it! ☐

Bubble sort

Data must be sorted into order to make it easier to understand and to use. There are many different sorting algorithms. The bubble sort algorithm compares adjacent data items and orders them. Several passes may be needed to sort the whole list.

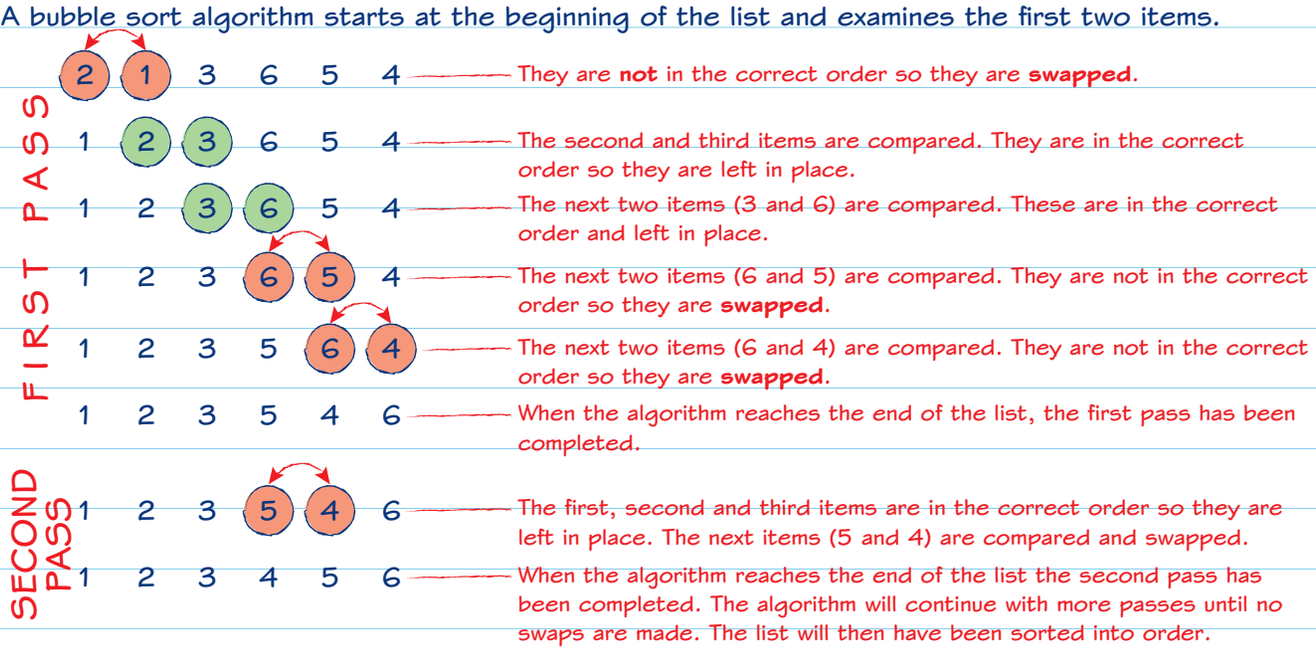
Order of sorting

Data can be sorted into:

- **ascending** order: 1, 2, 3, 4, 5 or a, b, c, d
- **descending** order: 5, 4, 3, 2, 1 or d, c, b, a

How bubble sort works

A bubble sort algorithm starts at the beginning of the list and examines the first two items.



Worked example

A student has stored the names of some friends in a list.

David	Claire	Stefan	Sophie	Matt

Show the stages of a bubble sort when applied to this data to sort it into ascending order. **(5 marks)**

Pass 1
Claire, David, Stefan, Sophie, Matt
Claire, David, Stefan, Sophie, Matt
Claire, David, Sophie, Stefan, Matt
Claire, David, Sophie, Matt, Stefan
Pass 2
Claire, David, Matt, Sophie, Stefan
Pass 3
Claire, David, Matt, Sophie, Stefan

All the comparisons in Pass 1 are given even if there was no swap (such as David/Stefan). Names swapped are underlined. The names of the passes have also been given.

Now try this

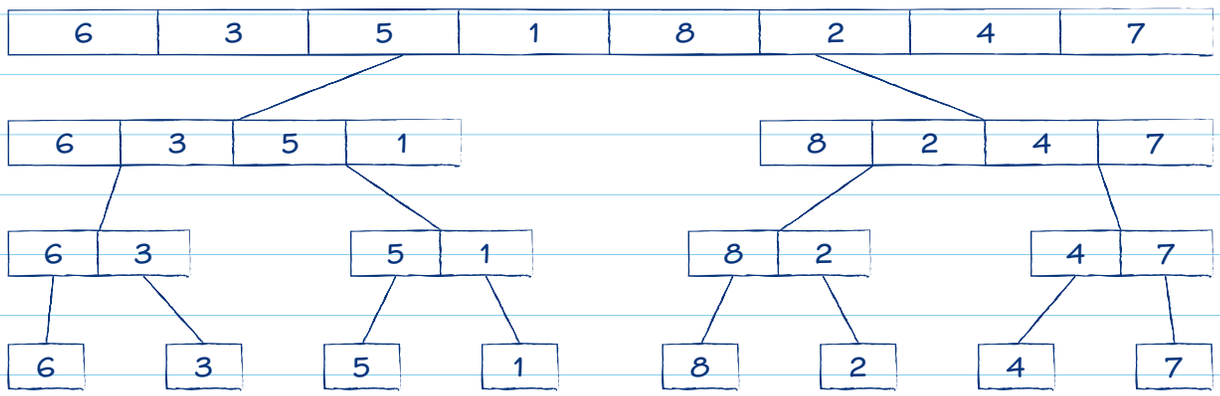
A list is made up of the numbers 4, 1, 2, 6, 3, 5.
Show the steps involved when sorting this list using a bubble sort algorithm. **(2 marks)**

Merge sort

The merge sort algorithm breaks a list into its component parts and then builds it up again with them in the correct order.

How merge sort works

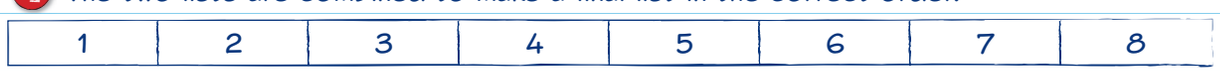
1 The algorithm breaks the list into two, and then these into two, over and over again.



2 The items are then reassembled in the same way but in ascending order.
3 is compared with 6, 5 is compared with 1, 2 is compared with 8, 4 is compared with 7 and they are placed in the correct order.

3 The leftmost items in each list are the lower items of those lists and the algorithm compares them – in this case, 3 with 1. The 1 is inserted in the new list and the 3 is then compared with the second number of the right-hand list (5). The 3 is inserted and the 5 is compared with the second number of the left-hand list (6). The same method is used for 2, 8, 4, 7 to form two lists.

4 The two lists are combined to make a final list in the correct order.



Worked example

Describe how data is sorted into ascending order using the merge sort algorithm. **(2 marks)**

The list is divided into two repeatedly until each list has only one item. The lists are then progressively merged with the items in ascending order.

For large numbers of items, a merge sort is far more efficient than a bubble sort as the problem is broken down into smaller and smaller ones, which are then easier to solve.

Now try this

A list is made up of the numbers 38, 27, 43, 3, 9, 82, 10.

Show the steps involved when sorting this list of numbers using a merge sort algorithm. **(5 marks)**

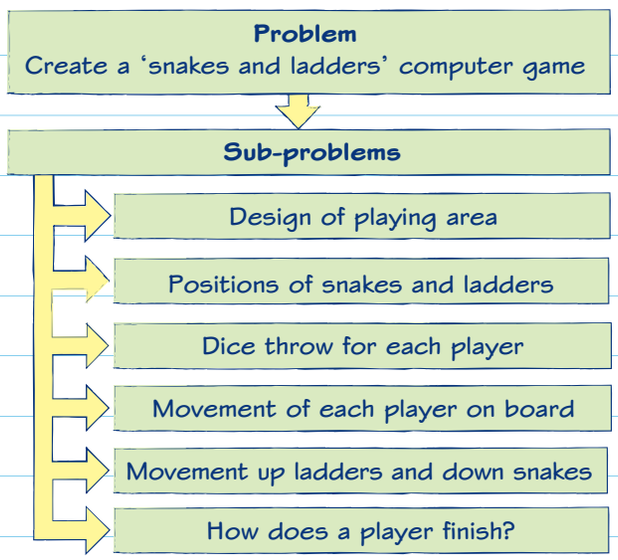
Had a look ☐ Nearly there ☐ Nailed it! ☐

Decomposition and abstraction

Computational thinking is a term used to describe the thought processes involved in understanding problems and formulating solutions in such a way that they can be carried out by computers.

Decomposition

Decomposition reduces a problem into sub-problems or components. These smaller parts are easier to understand and solve. This is an example of a divide-and-conquer approach.



Abstraction

Abstraction identifies essential elements that must be included in the computer models of real-life situations and discards inessential ones.

For a computer model of 'Snakes and Ladders', the sub-problem 'Dice throw for each player' includes the essential element: 'Generate a random number between 1 and 6'.

Inessential elements would include whether you use a shaker, how long you shake it for and how far you throw the dice.

Real life	Computer model
Dice throw for each player	Generate a random number between 1 and 6.
Move counter	Calculate new position.
Move up a ladder or down a snake	Calculate if user is in same position as start of a ladder or snake. Calculate new position.

Investigating requirements

All the requirements must be investigated when analysing a problem.

Inputs – all the data that will be entered by a user, e.g. how a user will indicate the movements and speed of a player in a football game.

Outputs – all the data that must be displayed to the users and how it will be displayed, e.g. as text or as images.

Processing – how the rules abstracted will be coded into the program, e.g. how the new position of a user will be calculated.

Initialisation – the values to which any variables must be set at the beginning of the program, e.g. SET score TO 0.

Worked example

In a computer 'Snakes and Ladders' game, there is a function named 'dice throw' which is called whenever it is a user's turn.

Give a reason why this function is an abstraction. **(1 mark)**

The 'dice throw' function is an abstraction because if focuses on the essential purpose of throwing a dice, i.e. to generate a number between 1 and 6, and ignores non-essential aspects of the problem, e.g. the colour of the dice or whether or not a shaker is used.

Abstraction and computer modelling

Computer programs can be used to model and simulate real-life situations such as weather and financial systems. If these are to provide meaningful results, then the rules that govern them must be abstracted accurately.

For example, in the financial model used by the Chancellor of the Exchequer:

- the actual effects of raising and lowering tax rates must be studied
- the rules are then abstracted for the computer model.

For example, if tax is lowered by a certain percentage then the percentage increase in employment and the sales of products must be abstracted accurately for the model.

Now try this

Describe what is meant by the following terms:

- decomposition
- abstraction. **(4 marks)**

Variables and constants

Variables and constants are used to store values in algorithms and programs. Variables **can change** while a program is running. Constants **must not change** while a program is running.

Identifiers

Variables and constants have names or **identifiers**.

Identifiers describe the data being stored.

- **Short** identifiers are easier to spell correctly each time they are used.
- **Long** identifiers can be used if they are more descriptive, e.g. `firstName`.

Identifiers cannot be the same as reserved words such as `print` or `while`.

Naming conventions

Variable and constant identifiers must be consistent throughout the program.

Camel case uses lower and upper case characters. (`FirstName` or `PricePerKilo`)

The first word can be lower case. (`firstName` or `pricePerKilo`)

In **snake case** the words are linked with an underscore. (`first_name` or `price_per_kilo`)

Variables and constants

Assignment

Values are assigned in **assignment statements** using the = symbol.

```
SET firstName TO 'David' or
SET pricePerKilo TO 30.
```

The variable value can be changed as the program is running, e.g.

```
SET pricePerKilo TO pricePerKilo * 2
```

would increase the value to 60.

Constants are assigned as `CONST REAL PI` or `SET PI TO 3.142`, for example, and used in calculations:

```
SET areaOfCircle TO PI * radius²
```

Assignments

An assignment is the association of a piece of data with a variable or constant, e.g. SET index TO 0. Often the assignment is done as an **input**, e.g.

```
RECEIVE name FROM (STRING)KEYBOARD
```

The value assigned to a variable can be **output** to the user, e.g.

```
SEND name TO DISPLAY
```

This will output the value stored in the variable `name` rather than outputting the word `'name'`.

Worked example

Identify the variables that are used in this algorithm and state the purpose of each.

```
RECEIVE length FROM (REAL) KEYBOARD
RECEIVE width FROM (REAL) KEYBOARD
SET area TO length * width
SEND area TO DISPLAY
```

(6 marks)

The variables are:

`length`, `width` and `area`.

`length` and `width` store values that are input by the user, and `area` stores the result of multiplying the length by the width.

Now try this

A student is writing an algorithm that allows a user to input their marks for five tests and then calculates the average mark.

Identify **two** variables from the scenario that need to be created to store the data in the program. **(2 marks)**

Arithmetic operators

Operators are used in algorithms to specify how values are to be manipulated.

Arithmetic operators in algorithms and programs

Arithmetic operators are used to perform calculations in algorithms and programs. The arithmetic operators in the Edexcel pseudo-code are shown below.

Operator	Function	Example
+	addition	`13 + 9 = 22` `SET totalScore TO score1 + score2 + score3`
-	subtraction	`24 - 11 = 13` `SET moneyLeft TO totalMoney - moneySpent`
*	multiplication	`6 * 9 = 54` `SET moneyTaken TO numberSold * unitPrice`
/	division	`36 / 5 = 5.2` `SET numberSold TO moneyTaken / unitPrice`
MOD	**Modulus** division Returns the **remainder** after the division of one number by another	`17 MOD 3` divides 17 by 3 and returns 2 as 17/3 = 5 with a remainder of 2. `SET remainder to number MOD 2`
DIV	**Quotient** division Returns the quotient or the lowest integer	`11 DIV 4 = 2` `SET completeHours TO minutes DIV 60`
^	Exponential powers of	`3 ^ 3 = 27` `SET area to Pi * radius ^ 2`

Order of operations

You need to use the correct order of operations in a calculation (BIDMAS).

Brackets
Indices or powers
Division
Multiplication
Addition
Subtraction

Using BIDMAS

The equation

```
number = 3^2 * 12 / (3*2) + 6
```

would be evaluated in this order:

3^2 * 12 / 6 + 6	brackets (3 × 2) = 6
9 * 12 / 6 + 6	index $3^2 = 9$
9 * 2 + 6	division 12 ÷ 6 = 2
18 + 6	multiplication 9 × 2 = 18
24	addition 18 + 6 = 24

Worked example

A chocolate factory packs 20 bars into each box. Write an algorithm to calculate and output the number of full boxes produced in a day. **(2 marks)**

```
RECEIVE numberOfBars FROM (INTEGER) KEYBOARD
SET numberOfBoxes TO numberOfBars DIV 20
SEND numberOfBoxes & ' full boxes have been produced.' TO DISPLAY
```

In the exam, you will be expected to apply your knowledge and use arithmetic operators in algorithms rather than being asked to evaluate equations.

In this example, the DIV operator has been used as the algorithm has to output the number of full boxes.

Now try this

Write an algorithm in pseudo-code that would allow a user to:

- input the number of items sold by a shop each day for 3 days
- calculate the total number of items sold
- calculate the mean number of items sold each day as an integer. **(3 marks)**

Relational operators

Relational operators are used to compare different items of data.

Relational operators

Operator	Function	Example

=	**Equal to**
	Checks if two values are equal.
		`IF length = width THEN`
		`  SEND 'It is a square.' TO DISPLAY`
		`END IF`

<>	**Not equal to**
	Checks if two values are not equal to each other.
		`IF length <> width THEN`
		`  SEND 'It is not a square.' TO DISPLAY`
		`END IF`

<	**Less than**
	Checks if one value is less than another.
		`IF myHeight < yourHeight THEN`
		`  SEND 'You are taller than me.' TO DISPLAY`
		`END IF`

<=	**Less than or equal to**
	Checks if one value is less than or equal to another.
		`IF myHeight <= yourHeight THEN`
		`  SEND 'I am taller than you.' TO DISPLAY`
		`END IF`

>	**Greater than**
	Checks if one value is greater than another.
		`IF myHeight > yourHeight THEN`
		`  SEND 'I am taller than you.' TO DISPLAY`
		`END IF`

>=	**Greater than or equal to**
	Checks if one value is greater than or equal to another.
		`IF score >= passMark THEN`
		`  SEND 'You have passed.' TO DISPLAY`
		`END IF`

Worked example

Write an algorithm in pseudo-code which asks the user to enter a number between 1 and 10 and then states if it is higher, lower or equal to 5. **(4 marks)**

```
RECEIVE number FROM (INTEGER) KEYBOARD
IF number > 5 THEN
  SEND 'Higher than 5.' TO DISPLAY
ELSE
  IF number < 5 THEN
    SEND 'Lower than 5.' TO DISPLAY
  ELSE
    SEND 'You entered 5.' TO DISPLAY
  END IF
END IF
```

Comparing strings

These operators can be used with strings as well as numbers. The strings are compared alphabetically.

For example,

`D` is higher in the alphabet than `C` so `David` is greater than `Catherine`.

`'Db'` is greater than `'Da'`.

Relational operators are used in **selection** using `IF...ELSE` statements to compare the values of variables. The `>` and `<` operators have been used but the `=` operator is not needed as the number must equal 5 if it is not less than nor greater than 5.

Now try this

Write an algorithm, in pseudo-code, that allows a user to enter two values as `value1` and `value2` and which will switch the values, if necessary, so that they are in ascending order.

(2 marks)

Logical operators

Logical operators are used to combine statements, or operands, which can be evaluated as true or false.

The AND operator

Using the AND operator ensures that the overall statement is true only if **all** of the individual statements are true.

```
IF number1 = 3 AND number2 = 6 AND
number3 = 10 THEN
  SET result TO True
END IF
```

For a result to be true they **all** have to be true – they have to be equal to 3, 6 and 10.

The OR operator

Using the OR operator ensures that the overall statement is true if **any** of the individual statements are true.

```
IF number1 = 3 OR number2 = 6 OR
number3 = 10 THEN
  SET result TO True
END IF
```

For a result to be true then any one or all of the statements has to be true, e.g. `IF number1 = 3` then the overall statement is true even if the other numbers are not 6 and 10.

The NOT operator

The NOT operator is used to reverse the logical state of the other operators.

```
IF NOT (number1 = 3 OR number2 = 6)
THEN
  SET result TO True
END IF
```

Without the NOT operator, the statement would be true if either `number1 = 3` **or** `number2 = 6`. Therefore, with the NOT operator, **both** must be false for the overall statement to be true. Note the use of brackets. The statement inside the brackets is evaluated before the condition outside is applied.

The OR statement in the algorithm is enclosed in brackets to make it easier for users to follow the logic and to determine the order in which the expressions are evaluated.

Worked example

State the output from this algorithm with these values. At the start of the algorithm X = 2, Y = 6 and Z = 9

```
IF (X = 3 OR Y = 6) AND NOT (Z=9) THEN
  SEND 'Conditions are met' TO DISPLAY
ELSE
  SEND 'Conditions are not met.' TO
  DISPLAY
END IF
```

(1 mark)

The output will be `'Conditions not met'`

The OR statement in brackets is true as Y is equal to 6.
But the overall statement is false as Z is equal to 9 and it must NOT be equal to 9 for the statement to be true.

Now try this

A student is writing a program that will search data to find suitable restaurants. The cost of the meal should be between £10 and £20. The type of food should be European or Asian and the restaurant should be no more than 10 miles away.

Complete the following algorithm by inserting the missing line.

```
RECEIVE cost FROM (REAL) KEYBOARD
RECEIVE type FROM (STRING) KEYBOARD
RECEIVE distance FROM (REAL) KEYBOARD

..................................................
  SEND 'This restaurant is suitable.' TO DISPLAY
ELSE
  SEND 'This restaurant is not suitable.' TO DISPLAY
END IF
```

(3 marks)

Selection

There are three basic programming constructs used to control program flow – sequence, selection and iteration. Sequence ensures that the commands are executed in the correct order and **selection functions** by choosing between two or more options. Selection can be represented in pseudo-code, flowcharts and written descriptions of algorithms.

Pseudo-code

IF/ELSE statements can be used if there are only two possible outcomes.

```
RECEIVE number FROM (INTEGER) KEYBOARD
IF number < 10 THEN
  SEND 'This is a low number.' TO
  DISPLAY
ELSE
  SEND 'This is a high number.' TO
  DISPLAY
END IF
```

The ELSE statement is used to state what should happen if the option in the IF statement is not true.

Most programming languages have an ELSEIF statement as these are more efficient. As soon as the correct condition is found, the rest of the options is not checked.

ELSE and IF

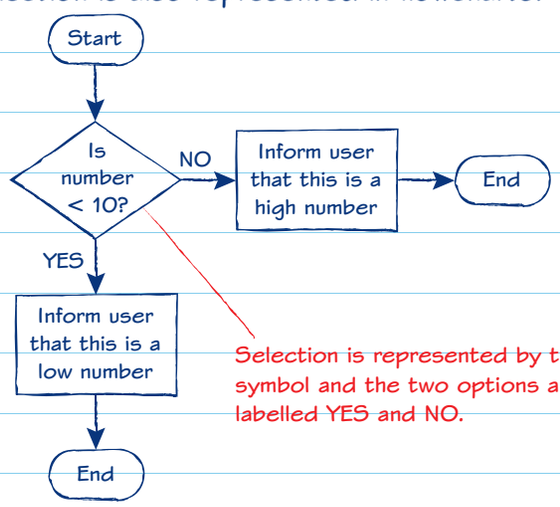
As there is no ELSEIF statement in the Edexcel pseudo-code, multiple selection is done by adding another IF construct under the ELSE statement. This is the method for checking the answer to a multiple-choice question.

```
RECEIVE answer FROM (character)
KEYBOARD
IF answer = 'A' OR answer = 'B' THEN
  SEND 'Sorry, that is incorrect.' TO
  DISPLAY
ELSE
  IF answer = 'C' THEN
    SEND 'Well done, that is correct.'
    TO DISPLAY
  ELSE
    SEND 'That is not recognised.' TO
    DISPLAY
  END IF
END IF
```

You must remember to include an 'END IF' for each IF statement.

Flowcharts

Selection is also represented in flowcharts.



Written descriptions

Usually the first stage in formulating an algorithm is a written description. For example:

Ask user to enter a number

If the number is less than 10 then inform the user that this is a low number.

If the number is greater than 10 then inform the user that this is a high number.

Worked example

A bowling alley gives a 5% reduction in charge if there are four or more people on a lane and a further 10% reduction if they are members. Write the code that will calculate the final charge. **(5 marks)**

```
IF numberOfPeople >= 4 THEN
  SET charge TO charge - (charge/100*5)
  IF membership = 'Yes' THEN
  SET charge TO charge -
  (charge/100*10)
  END IF
END IF
SEND 'The charge is ' & charge TO
DISPLAY
```

Now try this

A shop gives a discount of 10% for purchases of £200 and over, up to a total discount of £300. Write an algorithm, in pseudo-code, that would allow a user to calculate the discount. **(4 marks)**

Iteration

Iteration is the process of repeating a set of instructions for a fixed number of times or until there is a desired outcome. It is executed by program constructs called loops. Iteration is sometimes referred to as repetition.

Count controlled iteration

Count controlled iteration is used when the number of iterations is known before the loop is started.

FOR loops

FOR loops instruct the loop to be executed **for** a set number of times, e.g.

```
FOR turns FROM 0 TO 3 DO
  SEND turns TO DISPLAY
END FOR
```

This would output 0, 1, 2, 3

FOR loop with a STEP

A step can be defined so that all the numbers are not used.

```
FOR index FROM 2 TO 10 STEP 2 DO
  SEND index TO DISPLAY
END FOR
```

The numbers 2, 4, 6, 8 and 10 would be displayed.

FOR EACH

This construct is useful for traversing a string or an array.

```
SET scores TO [9, 6, 7, 3, 9]
FOR EACH score FROM scores DO
  SEND score TO DISPLAY
END FOR
```

Condition controlled iteration

Condition controlled iteration is used when the number of times a loop is executed is determined by a condition.

WHILE loops

WHILE loops continue while a condition remains true and stop when it becomes false.

```
SET number TO 0
WHILE number <> 3 DO
  RECEIVE number FROM (INTEGER) KEYBOARD
END WHILE
```

The loop will continue while the number does not equal 3. As soon as the number is equal to 3, the condition becomes false and the loop will stop.

REPEAT ... UNTIL loops

REPEAT ... UNTIL loops continue until a condition becomes true.

```
REPEAT
  RECEIVE number FROM (INTEGER) KEYBOARD
UNTIL number = 3
```

In a WHILE loop the condition is tested at the start of the loop and is called a pre-conditioned loop but in a DO ... UNTIL loop it is tested at the end and will therefore run at least once. It is called a post-conditioned loop.

Worked example

Write an algorithm in pseudo-code that will print out a times table for any number entered by a user. The table should run from times 1 to 12. **(4 marks)**

```
RECEIVE number FROM (INTEGER) KEYBOARD
FOR times FROM 1 TO 12 DO
  SEND number & ' x ' & times & ' = '
  & number * times TO DISPLAY
END FOR
```

In the SEND statement, variables and text have been **concatenated** (combined). They are joined by the '&' symbol and the text to be printed is enclosed in speech marks.

REPEAT ... TIMES

This loop will be repeated the number of times indicated in the expression.

```
REPEAT 10 TIMES
  RECEIVE score FROM (INTEGER) KEYBOARD
END REPEAT
```

Now try this

Write an algorithm in pseudo-code for a game that will generate a random number between 1 and 10 for a user to guess. The user has three guesses. If they guess correctly they are told they are correct. If they do not guess correctly, they are told the randomly generated number. **(6 marks)**